МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ

ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет информационных технологий

Кафедра информационных систем и технологий

Специальность Информационные системы и технологии

**ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №7 НА ТЕМУ:**

**Перемежение/деперемежение данных в информационно-вычислительных системах**

Выполнил студент 3 курса 1 группы

Халалеенко Андрей Николаевич

Минск 2024

**Задание 1:** Необходимо разработать авторское приложение в соответствии с целью лабораторной работы. По умолчанию используется блочный перемежитель/деперемежитель. По желанию студент может использовать иной. Задание выполняется по указанию преподавателя в соответствии с вариантом из таблицы.

**Задание 2:** Местоположение заданной группы ошибок выбирается (генерируется) случайным образом. Необходимо для группы ошибок каждой длины сгенерировать 30−40 случайных ситуаций. После деперемежения и исправления ошибок в сообщении сравнить передаваемую последовательность и полученную после исправления ошибок. Проанализировать эффективность перемежения/деперемежения.

Для выполнения задания был использован следующий код на языке *Typescripts*:

class Program {

    static Main(): void {

        const Xk: string = "101010101010101010101010101010101010101010101010101010101010101010101010101010101010101010101010101010101010";

        console.log(Xk.length.toString());

        const Xr: string = "10011";

        const kc: number = Xk.length;

        const rc: number = 4;

        const nc: number = 112;

        let masXk: number[] = [];

        this.StrInMas(masXk, Xk);

        let masXr: number[] = [];

        this.StrInMas(masXr, Xr);

        console.log("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_");

        console.log("Входная строка: " + Xk);

        console.log("Порождающий полином: " + Xr);

        console.log("k = " + kc + ", r = " + rc + ", n = " + nc);

        console.log("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_");

        let generationMatrix: number[][] = Array.from({ length: kc }, () => Array(nc).fill(0));

        this.CreateGenerationMatrix(generationMatrix, masXr, kc, nc);

        console.log("\nПорождающая матрица");

        this.OutMatrix(generationMatrix, kc, nc);

        this.CreateCanonicalMatrix(generationMatrix, kc, nc);

        console.log("\nКаноническая матрица");

        this.OutMatrix(generationMatrix, kc, nc);

        let checkMatrixc: number[][] = Array.from({ length: nc }, () => Array(rc).fill(0));

        this.CreateCheckMatrix(checkMatrixc, generationMatrix, kc, nc);

        console.log("\nПроверочная матрица");

        this.OutMatrix(checkMatrixc, nc, rc);

        //6.2

        let masXn: number[] = new Array(nc).fill(0);

        this.Shift(masXn, masXk, rc);

        //2.

        console.log("\nДеление");

        this.SearchResidue(masXn, masXr);

        console.log("Остаток (S-синдром):");

        this.OutMass(masXn);

        console.log("\n");

        console.log("Итоговая строка:");

        this.Shift(masXn, masXk, rc);

        this.OutMass(masXn);

        console.log();

        let save: number[] = [];

        let prop: number = 0;

        for (const menuitem of masXn) {

            save[prop++] = menuitem;

        }

        this.Alternation(masXn);

        console.log("\nСтрока после перемежения: ");

        this.OutMas(masXn);

        let error: number;

        let errorLenght: number;

        try {

            console.log("\n\"Место ошибки 2");

            error = 2;

            console.log("Длина ошибки 3");

            errorLenght = 3;

            for (let i = error; i < (error + errorLenght); i++) {

                masXn[i] = (masXn[i] + 1) % 2;

            }

        }

        catch { }

        console.log("\nСтрока с ошибками: ");

        this.OutMas(masXn);

        this.ReAlternation(masXn);

        console.log("\nСтрока после re:перемежения: ");

        this.OutMas(masXn);

        this.SearchError(masXn, masXk, checkMatrixc, rc);

        console.log("\n\nСтрока после исправления ошибок: ");

        this.OutMas(save);

        let masK: number[] = new Array(90).fill(0);

        let masK2: number[] = new Array(90).fill(0);

        //RemoveCheckBits(masK2, masXn, checkMatrixc);

        console.log("\n\nСтрока после удаления проверочных бит: ");

        console.log(Xk);

        console.log("");

        this.OutMassive(Xk);

    }

    static Alternation(masN: number[]): number[] {

        let matrix: number[][] = Array.from({ length: 16 }, () => Array(7).fill(0));

        //Получение матрицы

        for (let i = 0, m = 0; i < 16; i++) {

            for (let j = 0; j < 7; j++, m++) {

                matrix[i][j] = masN[m];

            }

        }

        console.log("\n\nПолученая матрица");

        this.OutMatrix(matrix, 16, 7);

        //Перемежение

        for (let i = 0, m = 0; i < 7; i++) {

            for (let j = 0; j < 16; j++, m++) {

                masN[m] = matrix[j][i];

            }

        }

        return masN;

    }

    static ReAlternation(masN: number[]): number[] {

        let k: number = 16;

        let n: number = 7;

        let matrix: number[][] = Array.from({ length: k }, () => Array(n).fill(0));

        //Получение матрицы

        for (let j = 0, m = 0; j < n; j++) {

            for (let i = 0; i < k; i++, m++) {

                matrix[i][j] = masN[m];

            }

        }

        console.log("\n\nПолученая матрица");

        this.OutMatrix(matrix, k, n);

        //RE:Перемежение

        for (let j = 0, m = 0; j < k; j++) {

            for (let i = 0; i < n; i++, m++) {

                masN[m] = matrix[j][i];

            }

        }

        return masN;

    }

    //Поиск синдрома

    static Sindrom(CheckMatrix: number[][], mas: number[], k: number): number[] {

        let r: number = this.HemmingLength(k);

        let n: number = r + k;

        let sindrom: number[] = new Array(r).fill(0);

        for (let i = 0, l = 0; i < r; i++, l = 0) {

            for (let j = 0; j < k; j++) {

                if (CheckMatrix[j][i] === 1 && mas[j] === 1) l++;

                else sindrom[i] = 0;

            }

            if (l % 2 === 1) sindrom[i] = 1;

            else sindrom[i] = 0;

        }

        for (let i = 0; i < r; i++) {

            mas[i + k] = sindrom[i];

        }

        return mas;

    }

    //Считаем r (кол-во пров. симв.)

    static HemmingLength(k: number): number {

        let r: number = Math.floor(Math.log2(k) + 1.99);

        return r;

    }

    static OutMas(mas: number[]): void {

        for (let i = 0; i < mas.length; i++) {

            process.stdout.write(mas[i].toString());

        }

    }

    //вывод матрицы

    static OutMatrix(matrix: number[][], k: number, n: number): void {

        for (let i = 0; i < k; i++) {

            for (let j = 0; j < n; j++) {

                process.stdout.write(matrix[i][j].toString());

            }

            console.log();

        }

    }

    static OutMassive(Xk: string): void {

        console.log(Xk);

    }

    //вывод одномерного массива

    static OutMass(mas: number[]): void {

        console.log();

        for (let i = 0; i < mas.length; i++) {

            process.stdout.write(mas[i].toString());

        }

        console.log();

    }

    static CreateGenerationMatrix(generationMatrix: number[][], mas: number[], k: number, n: number): number[][] {

        //Заполняем первую строку в проверочной матрице

        for (let i = 0; i < n; i++) {

            if (i < mas.length) {

                generationMatrix[0][i] = mas[i];

            }

            else {

                generationMatrix[0][i] = 0;

            }

        }

        //Сдвигаем каждую строки вправо от предыдущей

        for (let i = 1; i < k; i++) {

            for (let j = 0; j < n - 1; j++) {

                generationMatrix[i][j + 1] = generationMatrix[i - 1][j];

            }

            generationMatrix[i][0] = generationMatrix[i - 1][n - 1];

        }

        return generationMatrix;

    }

    //Приведение порождающей матрицы к каноническому виду

    static CreateCanonicalMatrix(generationMatrix: number[][], k: number, n: number): number[][] {

        //Перебираем строки для преведению к каноническому виду

        for (let i = 0; i < k; i++) {

            let i2 = i + 1;

            //Перебираем элементы строки, но только до k-элемента

            for (let j = i + 1; j < k; j++) {

                //если мы нашли единицу в строке, то...

                if (generationMatrix[i][j] === 1) {

                    //перебираем этот столбец, пока не найдем единицу

                    for (; i2 < k; i2++) {

                        let repeat = false;

                        //Если нашли, то складываем обе строки

                        if (generationMatrix[i2][j] === 1) {

                            for (let j2 = j - 1; j2 > 0; j2--) {

                                //Проверяем, есть ли до этой 1 еще 1, если есть то эту строку пропускаем

                                if (generationMatrix[i2][j2] === 1) {

                                    repeat = true;

                                }

                            }

                            if (repeat) continue;

                            this.AddingLinesMatrixMod2(generationMatrix, i, i2, n);

                            i2++;

                            break;

                        }

                    }

                }

            }

        }

        return generationMatrix;

    }

    //Преобразование канонической матрицы в проверочную

    static CreateCheckMatrix(checkMatrix: number[][], generationMatrix: number[][], k: number, n: number): number[][] {

        const r: number = n - k;

        for (let i = 0; i < k; i++) {

            for (let j = 0; j < r; j++) {

                checkMatrix[i][j] = generationMatrix[i][k + j];

            }

        }

        for (let i = k; i < n; i++) {

            for (let j = 0; j < r; j++) {

                if (j === i - k) {

                    checkMatrix[i][j] = 1;

                }

                else {

                    checkMatrix[i][j] = 0;

                }

            }

        }

        return checkMatrix;

    }

    //Сложение строк матрицы

    static AddingLinesMatrixMod2(matrix: number[][], str1: number, str2: number, lengthString: number): number[][] {

        for (let i = 0; i < lengthString; i++) {

            matrix[str1][i] = (matrix[str1][i] + matrix[str2][i]) % 2;

        }

        return matrix;

    }

    //Смещение на массива r

    static Shift(shiftMas: number[], mas: number[], r: number): number[] {

        for (let i = 0; i < mas.length; i++) {

            shiftMas[i] = mas[i];

        }

        return shiftMas;

    }

    //Преобразование сторки в массив

    static StrInMas(mas: number[], str: string): number[] {

        for (let i = 0; i < str.length; i++) {

            if (str[i] === '1')

                mas[i] = 1;

            else

                mas[i] = 0;

        }

        return mas;

    }

    static SearchError(masXn: number[], masXk: number[], checkMatrixc: number[][], r: number): number[] {

        const n: number = masXn.length;

        const k: number = n - r;

        let masXnSecond: number[] = new Array(n).fill(0);

        for (let i = 0; i < n; i++) {

            masXnSecond[i] = masXn[i];

        }

        console.log("\nДеление");

        this.SearchResidue(masXnSecond, masXk);

        console.log("\n");

        console.log("\nОстаток:");

        this.OutMass(masXnSecond);

        for (let i = 0; i < n; i++) {

            let coincidence = 0;

            for (let j = 0; j < r; j++) {

                if (checkMatrixc[i][j] === masXnSecond[k + j]) {

                    coincidence++;

                }

            }

            if (coincidence === r) {

                masXn[i] = (masXn[i] + 1) % 2;

                break;

            }

        }

        return masXn;

    }

    static SearchResidue(masXn: number[], masXr: number[]): number[] {

        const end: number = masXn.length - masXr.length + 1;

        for (let i = 0; i < end; i++) {

            if (masXn[i] === 1) {

                this.AddingMasMod2(masXn, masXr, i);

                this.OutMass(masXn);

            }

        }

        console.log("\n");

        return masXn;

    }

    //Сложение массивов по модулю 2 с опр. позиции

    static AddingMasMod2(mas1: number[], mas2: number[], pos: number): number[] {

        const end: number = pos + mas2.length;

        for (let i = pos; i < end; i++) {

            mas1[i] = (mas1[i] + mas2[i - pos]) % 2;

        }

        return mas1;

    }

}

Program.Main();

Листинг 1 – задания

Результат работы итогового кода:

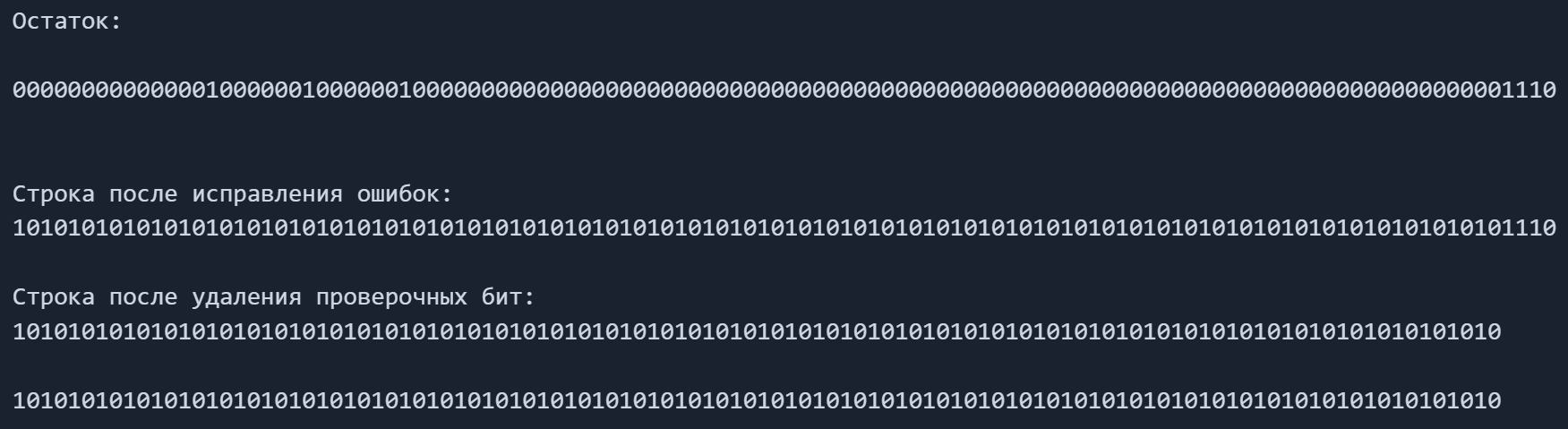
 

Рисунок 1 ­­­­– Результат работы программы

С увеличением длины пакета ошибок снижается эффективность перемежения/деперемежения.

**Вывод:** в ходе лабораторной работы были приобретены практические навыки использования методов перемежения/деперемежения двоичных данных в информационных системах.